

APPENDIX

Computer-Implemented Image Acquisition System

U.S. Patent Application Serial No. 09/152,432, filed September 14, 1998

U.S. Patent Application Serial No. 09/152,432, filed September 14, 1998

TECHNICAL FIELD

This invention relates to computer-implemented systems for managing imaging devices, such as digital cameras, scanners, and the like. This invention also relates to graphical window user interfaces, and particularly to user interfaces used to facilitate capture and storage management of digital images. This invention further relates to operating systems and browsers that incorporate image device managers and user interfaces.

BACKGROUND

Digital imaging devices, such as scanners, cameras, video cameras, have been experiencing rapid growth in popularity as their price tags continue to decrease. Recreational photographers enjoy capturing pictures and videos and placing the digital files onto their computers for printing or emailing to friends and relatives. Businesses use scanners to digitally record documents used in day-to-day operation for archival purposes.

Other solutions to this problem already exist. For example, TWAIN and ISIS are two image acquisition systems that are available today. However, both of these solutions have problems. TWAIN lacks robustness and interoperability. ISIS is a proprietary design that renders it difficult to use with other applications.

Accordingly, a task set before the inventor was to create an image acquisition system that was based on an open architecture model and could be integrated with existing applications and operating systems to provide a convenient environment for the user.

SUMMARY

This invention concerns an image acquisition system that offers an open architecture to integration with existing operating systems and other applications.

In an exemplary implementation, the image acquisition system is implemented on computer, such as a desktop personal computer, having a processing unit, memory, and operating system. One or more imaging devices are coupled to the computer. Examples of the imaging devices include a scanner, a digital camera, a digital video camera, and so forth. Some imaging devices, such as digital cameras, have a device memory and are capable of capturing a digital image and storing the image on its memory. Other imaging devices, such as scanners, may not have their own device memory.

The image acquisition system further includes an image device manager that is implemented in software on the computer to control operation of the imaging devices. The image acquisition system presents a user interface (UI) within the familiar graphical windowing environment. The UI presents a graphical window having a context space that pertains to a particular imaging context (e.g., scanning, photography, and video). In the camera context, the context space presents image files stored on the camera memory and/or on the computer memory. In the scanner context, the context space includes a preview scan area that reveals a preview of the image in the scanner. In the video context, the context space presents video clips stored on the computer memory, but logically represented as belonging to the video camera.

The UI also has a persistently visible imaging menu positioned within the context space that lists options particular to an imaging context. For example, if the context space pertains to the camera context, the menu lists options to take a picture, store a captured image on the computer, send the image in an email, and so on. In the scanner

context, the menu lists options to select an image type, preview an image, send the image to a particular destination, and scan the image.

The image acquisition system also includes a set of application program interfaces (APIs) that expose image management functionality to applications. The APIs enable applications to manage loading and unloading of imaging devices, monitor device events, query device information properties, create device objects, capture images using the devices, and store or manipulate the images after their capture.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a block diagram of an image acquisition system.

Fig. 2 is a block diagram of a software architecture for the image acquisition system.

Fig. 3 is a diagrammatic illustration of a graphical user interface window showing integration of the image acquisition system within a familiar file system setting.

Fig. 4 is a diagrammatic illustration of a graphical user interface window showing an opening window for managing imaging devices.

Fig. 5 is a diagrammatic illustration of a graphical user interface window for interfacing with a scanner.

Fig. 6 is a diagrammatic illustration of a graphical user interface window for interfacing with a digital camera.

Fig. 7 is a diagrammatic illustration of a graphical user interface window for interfacing with a digital video camera.

BRIEF DESCRIPTION OF THE APPENDIX

1 An attached appendix forms part of this document. The appendix contains a
2 description of methods implemented in an image acquisition API (application program
3 interface) utilized by the image acquisition system.

4 5 **DETAILED DESCRIPTION**

6 This invention concerns a computer-implemented image acquisition system that
7 manages imaging devices (such as digital cameras, scanners, digital video cameras, and
8 the like) and the images captured by them. In a preferred implementation, the image
9 acquisition system is implemented in a general-purpose computer (e.g., personal
10 computer, laptop, etc.) to manage imaging devices attached locally to the computer or
11 coupled remotely via a network. The image acquisition system supports a graphical user
12 interface windowing environment that integrates image device management with the
13 same look and feel of familiar browsing user interfaces for conventional file systems. In
14 this manner, a user encounters a familiar experience when managing the imaging devices
15 and image files from his/her computer.

16 17 **Exemplary System Architecture**

18 Fig. 1 shows an image acquisition system 20 having a computer 22 coupled to
19 multiple imaging devices 24-30. The computer 22 is a general-purpose computing device
20 that is described and illustrated, for discussion purposes, as a desktop personal computer
21 (PC). The computer 22 has a processing unit 40, a volatile memory 42 (e.g., RAM), a
22 non-volatile data memory 44 (e.g., disk drive, etc.), a non-volatile program memory 94
23 (e.g., ROM, disk drive, CD-ROM, etc.), a display 48 (e.g., VGA monitor), and a
24 universal serial bus (USB) 50. An operating system/browser 52 is stored in program
25

memory 46 and executed on the processing unit 40 when the computer is booted. Examples of suitable operating systems 52 include the Windows-brand operating systems from Microsoft Corporation and the operating systems from Apple Computer. Although a USB 50 is shown and described, other bus architectures may be used, including general serial buses, a SCSI bus, an IEEE 1394 serial bus that conforms to the IEEE 1394 specification, and so forth.

The imaging devices 24-30 are coupled to the computer via a serial connection to the USB 50. Illustrated examples of the imaging devices include a scanner 24, a video camera 26, and a digital camera 28. However, other imaging devices (e.g., copiers, facsimile machines, etc.) may also be used in conjunction with aspects of this invention, as represented by the generic imaging device 30. Some of the imaging devices have their own memory, as represented by memory 32 in imaging device 30. For example, the digital camera may have its own memory, whereas the scanner typically does not have a memory.

The image acquisition system 20 includes an image device manager 60, which is implemented as a software component loaded on the computer 22. More particularly, the image device manager 60 is stored in program memory 46 and runs on processing unit 40 during execution. The image device manager 60 may be integrated into the operating system 52 (as shown), executed as a set of services, or implemented as a separate self-contained program.

The image acquisition system 20 also has a user interface 62, which is preferably a graphical user interface that presents a graphical window having a context space pertaining to the imaging context. Depending upon the particular context, the context space may list available imaging devices for which device drivers have been loaded onto

the computer, or list digital image files captured by one or more of the imaging devices, or show an image being scanned in by the scanner.

The user interface 62 further presents a persistently visible imaging menu positioned within the context space. The imaging menu lists options that are particular to controlling the various imaging devices. For instance, when the context space pertains to the camera context, the menu lists a “Take Picture” option that is specific to operating the digital camera 28. Upon user selection of “Take Picture”, the image device manager 60 directs the digital camera 28 to record the current image obtained through the camera lens. In the scanner context, the menu lists a “Scan/Open” option that is particular to operating the scanner. Upon selection of this option, the image device manager 60 directs the scanner to scan the current image.

The image device manager 60 has an image device driver 64 and a set of APIs (application program interfaces) 66. The image device driver 64 controls operation of the imaging device in response to selected options in the context-specific menu. The driver 64 is the code that facilitates communication with the imaging device over the USB 50 and passes commands to capture an image, to read image files from the device’s local memory, to obtain the device’s properties, and so forth.

The APIs 66 define a set of interfaces that can be used to access the functionality of the image device manager 60. These APIs are described in detail in an Appendix to this disclosure, which is incorporated herein.

Exemplary Software Architecture

Fig. 2 shows a software architecture 70 for implementing the image acquisition system. At the kernel level, the architecture 70 includes kernel I/O drivers that include a bus driver to drive serial communication with the imaging device over the USB 50.

At the user level, a device driver 74 is loaded for the particular imaging device connected to the computer. The device driver 74 includes a device object, an optional UI, and optional image processing capabilities. An image device manager object 76 is called to initialize and select an image device, and create the device interface. The image device manager object 76 performs such tasks as instantiating a device driver object 74, determining the device status, monitoring events from the device, and so forth.

A COM (component object model) layer 78 exposes the device driver object 74 and image device manager object 76 to an upper level application 80. The application layer 80 represents both traditional TWAIN based applications that utilize a TWAIN compatibility layer 82, as well as new applications that support the APIs 66. Unlike the traditional TWAIN model, however, the TWAIN compatibility layer 82 interacts with the COM-based objects 74 and 76 rather than TWAIN-based devices.

Image Acquisition User Interface

The image acquisition system may be incorporated into the operating system, exist as a set of services, or be run as a separate, self-contained application. For discussion purposes, the image acquisition system is described as being integrated into an operating system that supports a graphical user interface windowing environment.

Fig. 3 shows an initial graphical user interface window 100 presented on the computer display 48. This window 100 is illustrated as the familiar “My Computer” screen within a browser-based windowing setting, which is well known to users of

Windows-brand operating systems. The “My Computer” window 100 presents a context for listing the major components that make up the user’s PC, including disk drives, printers, a control panel, and networking functionality.

Of interest to the image acquisition system is the integration and treatment of the imaging devices as a folder 102 organized with the other general computer components. This provides a convenient starting point for the user to access the imaging devices 24-30 that are coupled to the computer 22.

When the user activates the “Imaging Devices” folder icon 102 for the first time, an installation Wizard comes up to guide the user through the installation of an imaging device. Suppose, for example, the user has installed two scanning devices and a digital camera. Activating the “Imaging Devices” icon 102 navigates to a new “Imaging Devices” window.

Fig. 4 shows the “Imaging Devices” window 110 presented on the computer display 48. The “Imaging Devices” window 110 pertains to an imaging context and lists the imaging devices that have been installed on the computer. In this example, the window lists an “add imaging device” icon 112 and icons for the three installed devices: a “My Scanner” icon 114 for a locally installed scanner, a “My Camera” icon 116 for the installed camera, and a “Jake’s Scanner” icon 118 for remotely installed (via a network connection) scanner. Activation of the “add imaging device” icon 112 recalls the wizard to enable the user to install any additional imaging devices.

The “Imaging Devices” window 110 distinguishes between devices that are currently available and those that are not available (e.g., offline, physically removed, etc.). Devices that are not available are dimmed and the user has the option of

uninstalling them. In Fig. 4, the second scanner identified as “Jake’s Scanner” is not available and hence the icon 118 is dimmed.

Activating one of the imaging devices listed in window 110 causes the image acquisition system to present different windows exhibiting contexts that are specific to the selected imaging device. Within these device-oriented windows, the image acquisition system presents context-specific menus that contain items or options pertinent and relevant to the particular imaging device.

Fig. 5 shows a “My Scanner” window 120 that is presented upon selection of the “My Scanner” icon 114 in Fig. 4. The scanner window 120 presents a context space 122 that pertains to the scanning context. The context space 122 has a preview scan space 124 and a persistently-visible, context-specific menu 126 positioned adjacent the preview scan space within the graphical window 120.

The context-specific menu 126 is always visible in the scanner window 120. The menu 126 offers options that are tailored to operating the scanner attached to the computer or remotely coupled to the computer via a network. While some of the options may be included in a context menu (i.e., a menu that appears near the pointer following a right mouse click), the persistently-visible menu 126 lists operating specific options tailored to the scanner that are not included elsewhere in the user interface.

The menu 126 includes an image type selection 128 that has a pull-down list of various image types from which a user may select. A non-exhaustive list of image types includes color photograph, black and white photograph, color line art, black and white line art, and text. The image types included in the pull-down list 128 are specific to the device. Some imaging devices may not provide support for a given format and hence the format is omitted in that particular list.

1 A destination selection 130 has a pull-down list of various choices on what to do
2 with the scanned image. For instance, the list 130 might include using the image in an
3 application, faxing the image, printing the image, copying the image to a clipboard, and
4 saving the image in a file. The destination selection simplifies the output operation for
5 the user. For example, selection of a choice directly affects the acquisition parameters
6 and image quality without requiring the user to know what parameters to set.

7 The persistently-visible context-specific menu 126 also has a "New Preview"
8 command 132 that directs scanners to create a preview image of an image that is
9 currently in the scanning bed. The image is presented in the preview scan space 124.
10 When the image appears in the scan space 124, a preview control 134 is provided to
11 allow the user to select a region of the image for a final scan. In the illustrated
12 implementation, the control 134 is shown as a dashed rectangular box framing the
13 picture. The user can manipulate the box 134 to capture all or less than all of the image.
14 Upon selection of the region, the control can proportionally resize the image to reflect the
15 size of the scanner bed and automatically configure the scanner to make the appropriate
16 adjustments to capture the selected image portion.

17 The menu 126 includes a "Scan/Open" command 136 to direct the scanner to
18 capture the image. When this command is selected, the scanner scans the image in its
19 bed. Concurrently with this scanning action, the image progressively appears in the
20 preview scan space 124 to visually convey that the scanner is scanning the image. In one
21 implementation, the image is progressively displayed row-by-row from top to bottom of
22 the image.
23

24 The menu 126 includes a "Save" option 138, which directs the scanner to capture
25 the image as a file and store the file in the computer memory. The last listed option is a

“Send to” option 140, which allows the user to send the image to various locations (or applications) on the PC, such as for packaging in a facsimile or email.

Fig. 6 shows a “My Camera” window 150 that is presented upon selection of the “My Camera” icon 116 in Fig. 4. The camera window 150 presents a context space 152 that pertains to the camera context. The context space 152 has a file space 154 and a persistently-visible, context-specific menu 156 positioned adjacent the file space within the graphical window 150.

The context-specific menu 156 is always visible in the camera window 150 and offers options that are tailored to operating the digital camera 28 attached to the computer. While some of the options may be included in a context menu (i.e., a menu that appears near the pointer following a right mouse click), the persistently-visible menu 156 lists operating specific options tailored to the camera that are not included elsewhere in the user interface.

The menu 156 is illustrated as having two tabs: a pictures tab 158 and a camera tab 160. Table 1 contains the options and corresponding functions available on the pictures tab 158.

Table 1

<u>Option</u>	<u>Function</u>
Open	Opens a picture with a default registered application.
Save in “My Pictures” folder	Downloads the images from the camera and copies them to “My Pictures” directory on computer memory.
Zoom	Changes the window view and allow the user to select one picture at the time and zoom in/out of the picture once it’s copied

locally.

Send to	Allows the user to send the picture to various locations (or applications) on the PC. For example, the user may choose to “send” the picture to an “email recipient”.
Lock on Camera	Allows the user to lock a picture to prevent accidental deletion.
Delete from Camera	Allows the user to permanently remove the picture from the camera after a confirmation.
Rotate to the Right	Allows the user to rotate the picture 90 degrees to the right.
Rotate to the Left	Allows the user to rotate the picture 90 degrees to the left.
View Properties	Allows the user to view properties associated with the selected picture(s).

Table 2 contains the options and corresponding functions available on the camera tab 160.

Table 2

<u>Option</u>	<u>Function</u>
Take Picture	Triggers the camera to take a picture.
Copy all Pictures	Copies all the pictures to designated location on the PC.
Remove all Pictures	Deletes all pictures in the camera.
Share	Brings up a wizard for the local user to share the camera.
Initialize Memory Card	Enables user to initialize the storage card in the camera.
View Properties	Allows the user to view a summary of the camera properties.

The file space 154 lists files and/or folders that pertain to digital images taken by the digital camera. The files are the images themselves (e.g., JPG files) and the folders contain image files and/or other folders with image files in them.

The file space 154 presents the files that are currently stored on the camera. In this manner, the user can easily view the camera memory as if it were another memory of the computer. The UI allows easy integration of the camera control into the familiar windowing environment.

To add a picture to the file space, the user captures a picture using the “Take Picture” command in the camera menu 160. The picture then appears as a file in the file space 154. The user can then select the image file by clicking on the file and manipulating the picture using the commands on the pictures menu 158, such as “Rotate to the Left”, “Rotate to the Right”, “Zoom”, and “Send to”. The user can also save the image file to the computer memory using the command “Save in My Pictures folder”.

Fig. 7 shows a modified “My Camera” window 170 that supports dual-mode cameras (i.e., video and still). The modified window 170 is presented upon selection of the “My Camera” icon 116 in Fig. 4 and is similar to the window 150 of Fig. 6 in that it has a context space 172 with a file space 174 and a persistently-visible, context-specific menu 176. However, in this modified implementation, the context-specific menu 176 also has a video tab 178 to list options pertaining to operation of the video camera 26.

Notice also that one of the files in the file space 174 is a play-in-place video file 180. This play-in-place video file 180 can be actuated to play a video clip or stream within the small area depicted as box 180. That is, the static video icon in box 180 is replaced with a streaming video at the same location in the file space. Play-in-place

video files 180 were first introduced in Media Manager, a multimedia application available from Microsoft.

Table 3 contains the options and corresponding functions available on the video tab 178.

Table 3

<u>Option</u>	<u>Function</u>
Play	Plays back a video stream from the video camera.
Open	Opens a video file with a default application.
Capture Frame	Directs the video camera to record a single still-image frame.
Capture Video	Directs the video camera to record a video clip.
View Properties	Allows the user to view a summary of the video camera properties.

Other commands may be added to the menu. For instance, a “stop” command may be employed to halt the capture of live video.

Image Acquisition API

The image acquisition API 66 enables applications to manage loading and unloading of all imaging devices, monitor device events, query device information properties, create device objects, capture images using the devices, and store or manipulate the images after their capture.

The interfaces are accessible by high level languages (e.g., Visual Basic) as well as lower level ones (e.g., C, C++, etc.). COM is a suitable interface. In this context, each

device is exposed as a COM object, whereby the object provides a number of methods and properties associated with the imaging device.

As one exemplary implementation, there are three general objects: a device manager object, a camera object, and a scanner object. The objects are described generally below. A more detailed description of the objects and methods are provided in the Appendix to this disclosure. This Appendix is incorporated into the disclosure.

The device object contains device context and status information for a physical device. Once a device object is created for a physical device, the physical device controls what device properties are available and what values the properties may assume. There may be multiple device objects created for any physical device. However, a device object has exclusive access to a physical device before any operation (i.e., scan, take a picture, etc.) is performed. Exclusive access to a physical device is made available through a locking/unlocking mechanism.

The device manager is implemented as three objects that perform the following functions:

A CImageInDevMgr object is used to:

- Create a device enumerator object
- Create a device object when given a DeviceID
- Display UI to let a user choose a device object
- Display UI to both choose a device and acquire an image from the chosen device.

A CEnumImageInDevInfo object is used to:

- Enumerate all ImageIn devices on a system. For each device enumerated, a CImageInDevInfo object is returned.

A CImageInDevInfo object is used to:

- Query device information properties from the ImageIn device. One of the properties, Device ID, can be used by CImageInDevMgr to create a device object.

The camera object may expose the following functions:

- Open and close the device for communication

- Control the device
- Update and read device properties
- Update and read picture properties
- Download, remove, and upload pictures to device

The scanner object may expose the following functions:

- Open and close the device for communication
- Control the device
- Update and read device properties
- Set operation intent

Although the invention has been described in language specific to structural features and/or methodological steps, it is to be understood that the invention defined in the appended claims is not necessarily limited to the specific features or steps described. Rather, the specific features and steps are disclosed as preferred forms of implementing the claimed invention.

CLAIMS

1. An image acquisition system comprising:
 - a computer having a memory;
 - an imaging device coupled to the computer, the imaging device having a device memory, the imaging device capturing a digital image and storing the image on one of the computer memory or the device memory;
 - an image device manager executable on the computer to control operation of the imaging device; and
 - a user interface with a context space and a persistently-visible imaging menu positioned within the context space, the imaging menu listing options that are particular to controlling the imaging device and managing images captured by the imaging device.
2. An image acquisition system as recited in claim 1, wherein the imaging menu includes a capture option that directs the imaging device to capture an image and to store the image in the memory of the computer.
3. An image acquisition system as recited in claim 1, wherein the context space lists image files stored on at least one of the device memory or the computer memory.
4. An image acquisition system as recited in claim 1, wherein:
 - the imaging device is a scanner; and
 - the context space presented by the user interface contains a preview scan area that shows a preview of an image in the scanner.

1 5. An image acquisition system as recited in claim 1, wherein :
2 the imaging device is a scanner; and
3 the context space presented by the user interface contains a scan area that is
4 initially empty prior to a time when the scanner scans an image and then progressively
5 displays the image as the scanner scans the image.

6 6. An image acquisition system as recited in claim 1, wherein the context
7 space exhibits a play-in-place video file at a location, the play-in-place video file being
8 actuatable to play a video stream at the location in the context space.

9
10 7. An image acquisition system as recited in claim 1, further comprising an
11 operating system stored in the memory and executable on the computer, the image device
12 manager being incorporated into the operating system.

13
14 8. An image acquisition system as recited in claim 1, further comprising a
15 browser stored in the memory and executable on the computer, the image device manager
16 being incorporated into the browser.

17
18 9. In an image acquisition system having a computer and an imaging device
19 coupled to the computer, an image manager implemented in software stored in the
20 computer comprising:

21 a user interface presenting a context space of an imaging context and a
22 persistently-visible imaging menu positioned within the context space, the imaging menu
23 listing options that are particular to operating the imaging device and managing image
24 files; and
25

an image device driver to control operation of the imaging device in response to selected options in the imaging menu.

10. An image manager as recited in claim 9, wherein the imaging menu includes a capture option to capture an image from the imaging device, and the image device driver directs the imaging device to capture the image into an image file for storage on the imaging device or the computer in response to selection of the capture option.

11. An image manager as recited in claim 9, wherein the context space holds image files and the imaging menu includes an option to operate on one or more of the image files.

12. An image manager as recited in claim 9, wherein the context space exhibits a play-in-place video file at a location, the play-in-place video file being actuatable to play a video stream at the location in the context space.

13. An image manager as recited in claim 9, wherein the imaging device coupled to the computer is a scanner and the context space of the user interface pertains to a scanner context and contains an area that is initially empty prior to a time when the scanner scans an image and then progressively displays the image as the scanner scans the image.

14. An operating system embodied on a computer readable medium comprising an image manager as recited in claim 9.

1 15. A file system embodied on a computer readable medium comprising an
2 image manager as recited in claim 9.

3
4 16. A browser program embodied on a computer readable medium comprising
5 an image manager as recited in claim 9.

6
7 17. A user interface embodied on a computer-readable medium and
8 executable on a computer comprising:

9 a file space of a selectable context, the file space exhibiting one or more files
10 and/or folders pertaining to the context; and

11 a persistently-visible context-specific menu positioned within the file space
12 adjacent to the files and/or folders, the context-specific menu listing options that are
13 particular to the context of the file space.

14
15 18. A user interface as recited in claim 17, wherein the files comprise digital
16 image files and the folders comprise sets of digital image files.

17
18 19. A user interface as recited in claim 17, wherein the context of the file
19 space pertains to imaging devices, and the options listed in the context-specific menu
20 include a command to capture digital images using an imaging device coupled to the
21 computer.
22
23
24
25

1 20. A user interface as recited in claim 17, wherein the file space also exhibits
2 a play-in-place video file at a location, the play-in-place video file being actuatable to
3 play a video stream at the location in the file space.

4
5 21. A user interface as recited in claim 17, wherein the options in the context-
6 specific menu change in response to changing the context of the file space.

7
8 22. A file system embodied on a computer-readable medium incorporating the
9 user interface as recited in claim 17.

10
11 23. An operating system embodied on a computer-readable medium
12 incorporating the user interface as recited in claim 17.

13
14 24. A browser embodied on a computer-readable medium incorporating the
15 user interface as recited in claim 17.

16
17 25. For a computer-implemented scanning system having a scanner coupled to
18 a computer, a user interface comprising a graphical window having a preview scan space,
19 the preview scan space being initially empty prior to a time when the scanner scans an
20 image, the user interface progressively displaying the image within the preview scan
21 space to visually convey that the scanner is scanning the image.
22
23
24
25

1 26. A user interface as recited in claim 25, wherein the user interface
2 progressively displays the image row-by-row.

3
4 27. A user interface as recited in claim 25, wherein the user interface
5 progressively displays the image currently with the scanner scanning the image.

6
7 28. A user interface as recited in claim 25, further comprising a persistently-
8 visible menu positioned adjacent the preview scan space within the graphical window,
9 the menu containing options that are particular to operating the scanner.

10
11 29. A user interface as recited in claim 25, further comprising a destination list
12 that presents a user with choices on what to do with the scanned image.

13
14 30. A user interface as recited in claim 25, further comprising a control to
15 enable a user to select which portion of the image to scan in a final output.

16
17 31. A file system embodied on a computer-readable medium incorporating the
18 user interface as recited in claim 25.

19
20
21 32. An operating system embodied on a computer-readable medium
22 incorporating the user interface as recited in claim 25.

1 33. A browser embodied on a computer-readable medium incorporating the
2 user interface as recited in claim 25.

3
4 34. For a computer-implemented scanning system having a scanner coupled to
5 a computer, a user interface embodied on a computer-readable medium and executable on
6 the computer comprising:

7 a graphical window having a preview scan space to hold a preview of the image
8 to be scanned by the scanner; and

9 a control to enable a user to select a portion of the image to be scanned by the
10 scanner.

11
12 35. An operating system embodied on a computer-readable medium
13 incorporating the user interface as recited in claim 34.

14
15 36. A browser embodied on a computer-readable medium incorporating the
16 user interface as recited in claim 34.

17
18 37. An application program interface for an image acquisition system, the
19 application program interface being embodied on a computer-readable medium and
20 having methods for performing the following functions:

21 creating a device object for an imaging device;

22 displaying a user interface to enable a user to choose the device object;

23 displaying a user interface to enable the user to capture an image using the
24 imaging device; and
25

querying the imaging device for properties.

38. An application program interface for an image acquisition system, the application program interface being embodied on a computer-readable medium and having methods for performing the following functions:

opening and closing a camera for communication;

controlling the camera; and

reading properties associated with the camera.

reading properties associated with pictures taken by the camera; and

manipulating pictures stored in a memory of the camera.

39. An application program interface for an image acquisition system, the application program interface being embodied on a computer-readable medium and having methods for performing the following functions:

opening and closing a scanner for communication;

controlling the scanner; and

reading properties associated with the scanner.

40. A computer-implemented method for execution in a graphical user interface windowing environment, comprising the following steps:

presenting a set of one or more files and/or folders in a file space within a graphical window;

presenting a persistently-visible, context-specific menu within the file space adjacent to the files and/or folders; and

1 listing options in the context-specific menu that are particular to operating on the
2 files and/or folders in the file space.

3
4 41. A computer-implemented method as recited in claim 40 wherein the
5 presenting step comprises the step of presenting a set of digital image files.

6
7 42. A computer-implemented method as recited in claim 40 further
8 comprising the step of operating on the files and/or folders in response to selection of an
9 option in the context-specific menu.

10
11 43. A computer-implemented method as recited in claim 40 further
12 comprising the step of exhibiting a play-in-place video file at a location in the file space,
13 the play-in-place video file being actuatable to play a video stream at the location in the
14 file space.

15
16 44. A computer-implemented method as recited in claim 40 further
17 comprising the following steps:

18 presenting a new set of one or more files and/or folders in the file space; and
19
20 listing new options in the context-specific menu that are particular to operating on
21 the new set of files and/or folders.

22
23 45. For a computer-implemented scanning system having a scanner coupled to
24 a computer, a computer-implemented method for executing a scanning software
25

1 application in a graphical user interface windowing environment, comprising the
2 following steps:

3 presenting a preview scan space within a graphical window, the preview scan
4 space being initially empty; and

5 progressively displaying an image within the preview scan space to visually
6 convey that the scanner is scanning the image.

7
8 46. A computer-implemented method as recited in claim 45 wherein the
9 displaying step comprises the step of building the image row-by-row.

10
11 47. A computer-implemented method as recited in claim 45 wherein the
12 displaying step comprises the step of building the image simultaneously as the scanner
13 scans the image.

14
15 48. A computer-implemented method as recited in claim 45 further
16 comprising the following steps:

17 presenting a persistently-visible menu within the preview scan space within the
18 graphical window; and

19 listing options in the menu that are particular to operating the scanner.
20
21
22
23
24
25

49. For a computer-implemented scanning system having a scanner coupled to
a computer, a computer-implemented method for executing a scanning software
application in a graphical user interface windowing environment, comprising the
following steps:

presenting a preview scan space to hold a preview of the image to be scanned by
the scanner; and

enabling a user to select a portion of the image to be scanned by the scanner.

ABSTRACT

An image acquisition system has a computer and one or more imaging devices coupled to the computer. Each imaging device has a device memory and is capable of capturing a digital image and storing the image in its memory. An image device manager is implemented in software on the computer to control operation of the imaging devices. The image device manager presents a user interface (UI) within the familiar graphical windowing environment. The UI has a context space that pertains to a particular imaging context (e.g., scanning, photography, and video). The UI also has a persistently-visible imaging menu positioned within the context space that lists options particular to the imaging context. For example, if the context space pertains to the digital camera context, the menu lists options to take a picture, store the image on the computer, send the image in an email, and so on. In the scanner context, the menu lists options to select an image type, preview an image, send the image to a particular destination, and scan the image. The image acquisition system also includes a set of application program interfaces (APIs) that expose image management functionality to applications. The APIs enable applications to manage loading and unloading of imaging devices, monitor device events, query device information properties, create device objects, capture images using the devices, and store or manipulate the images after their capture.

Fig. 1

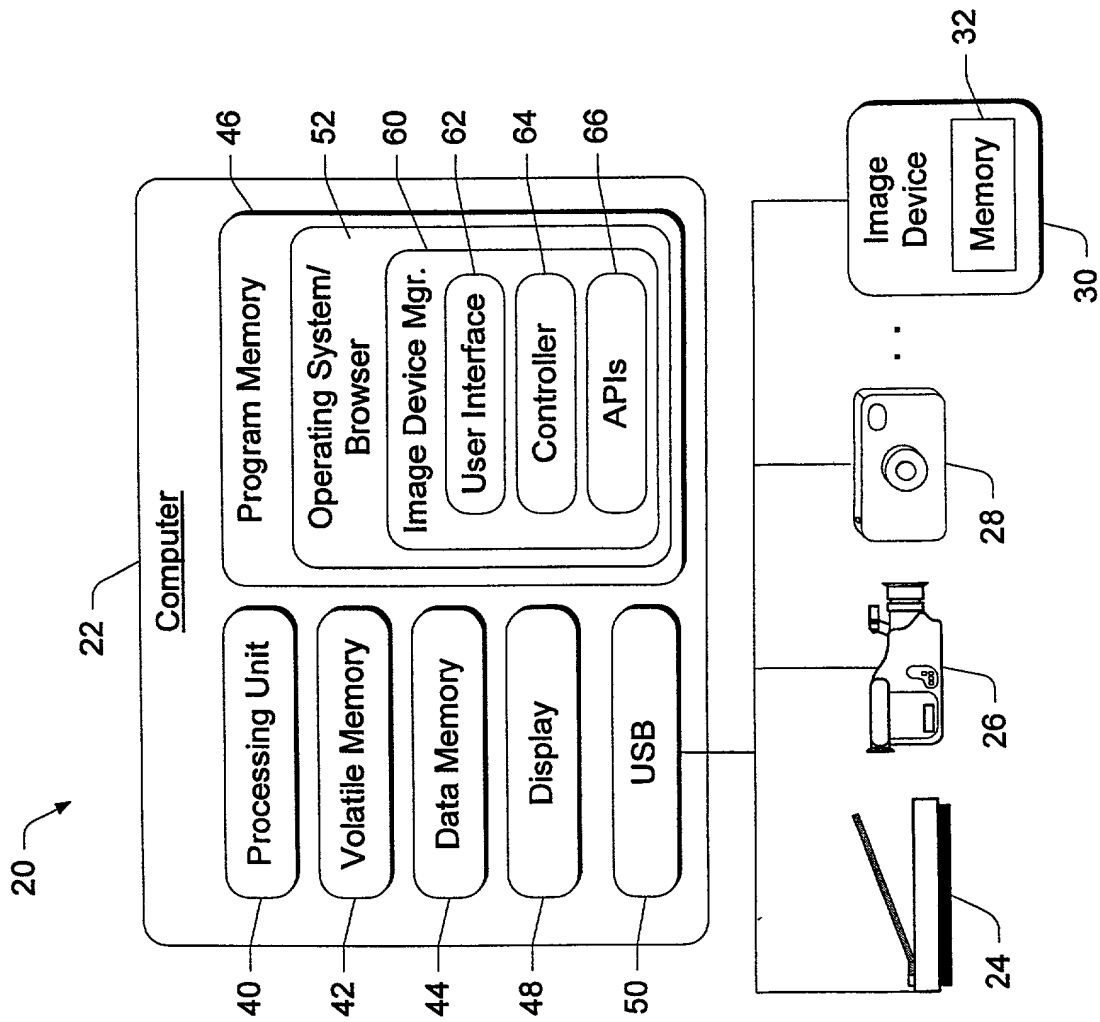


Fig. 2

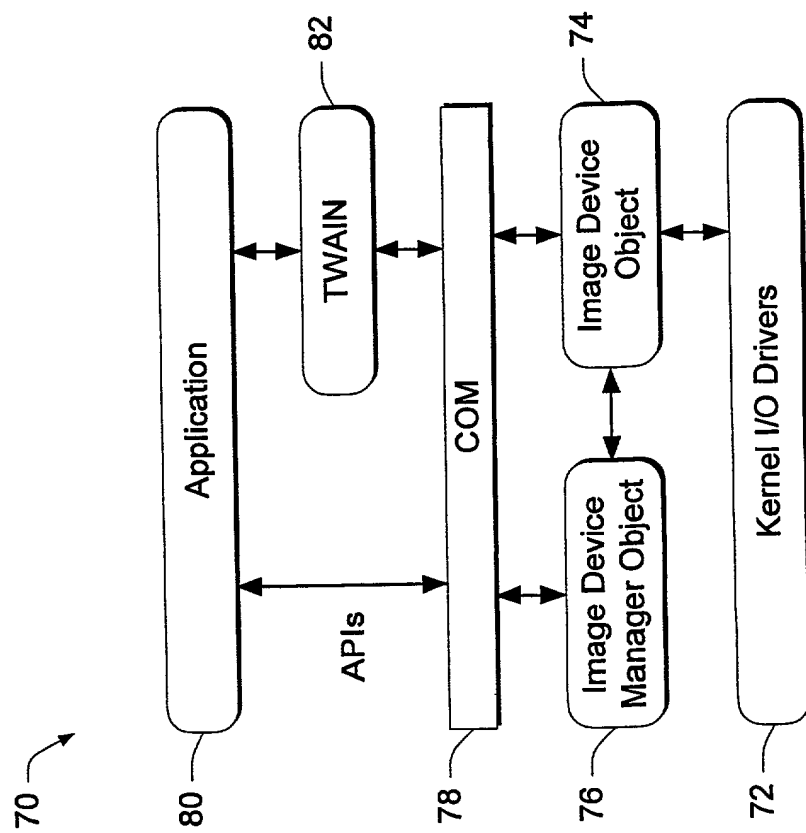


FIG. 3

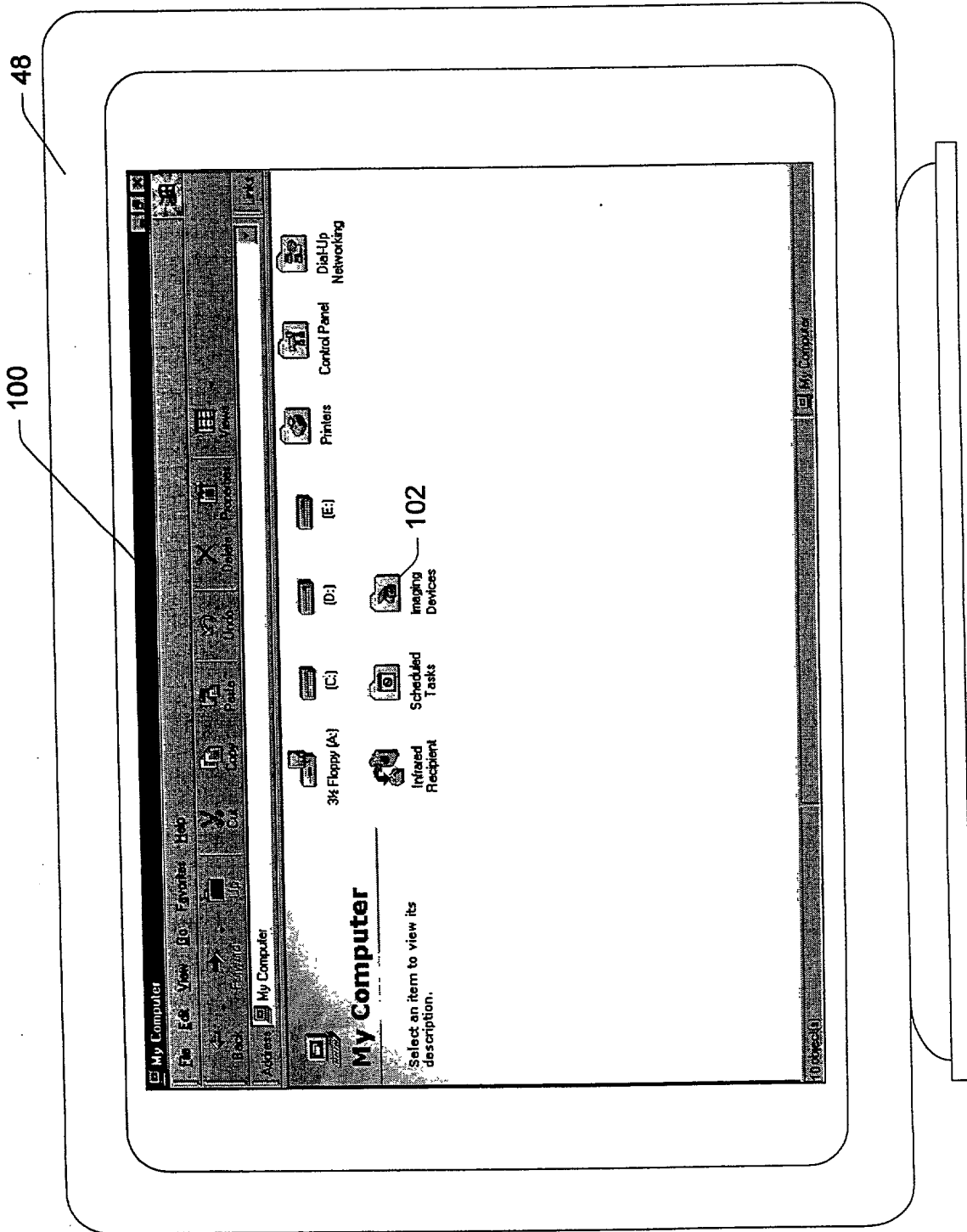


Fig. 3

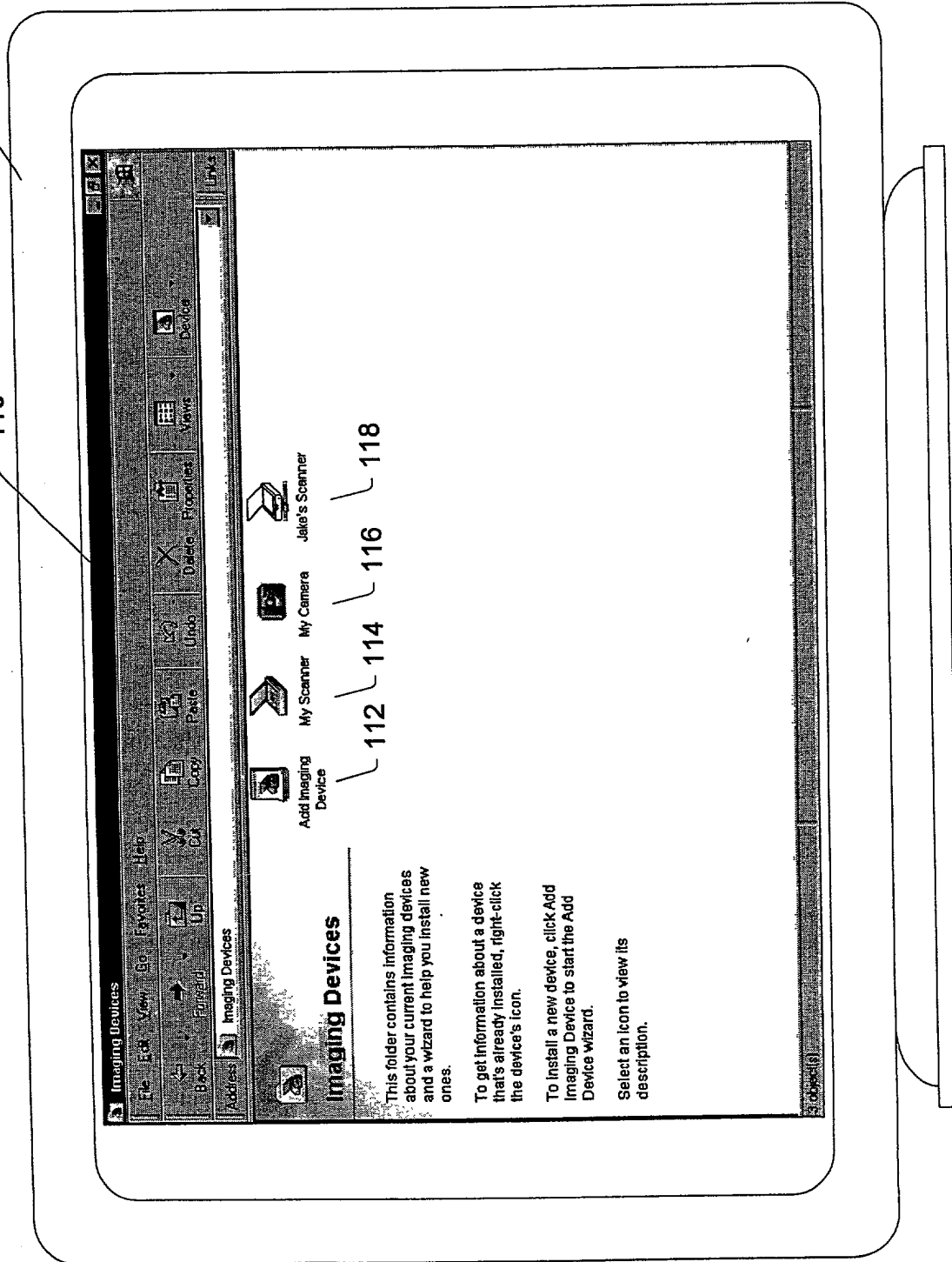


Fig. 4

FIG. 5

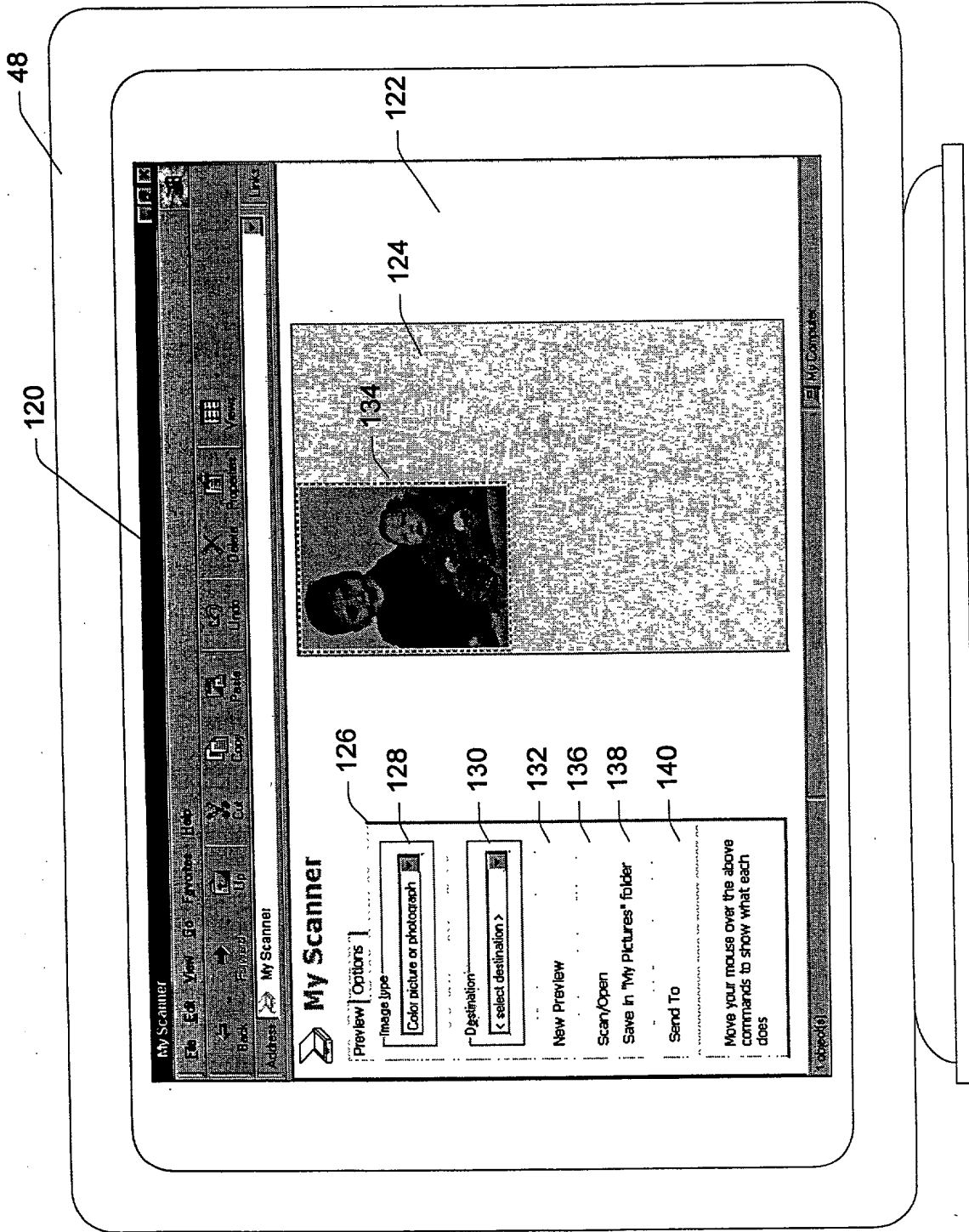


Fig. 5

10440-036-0360

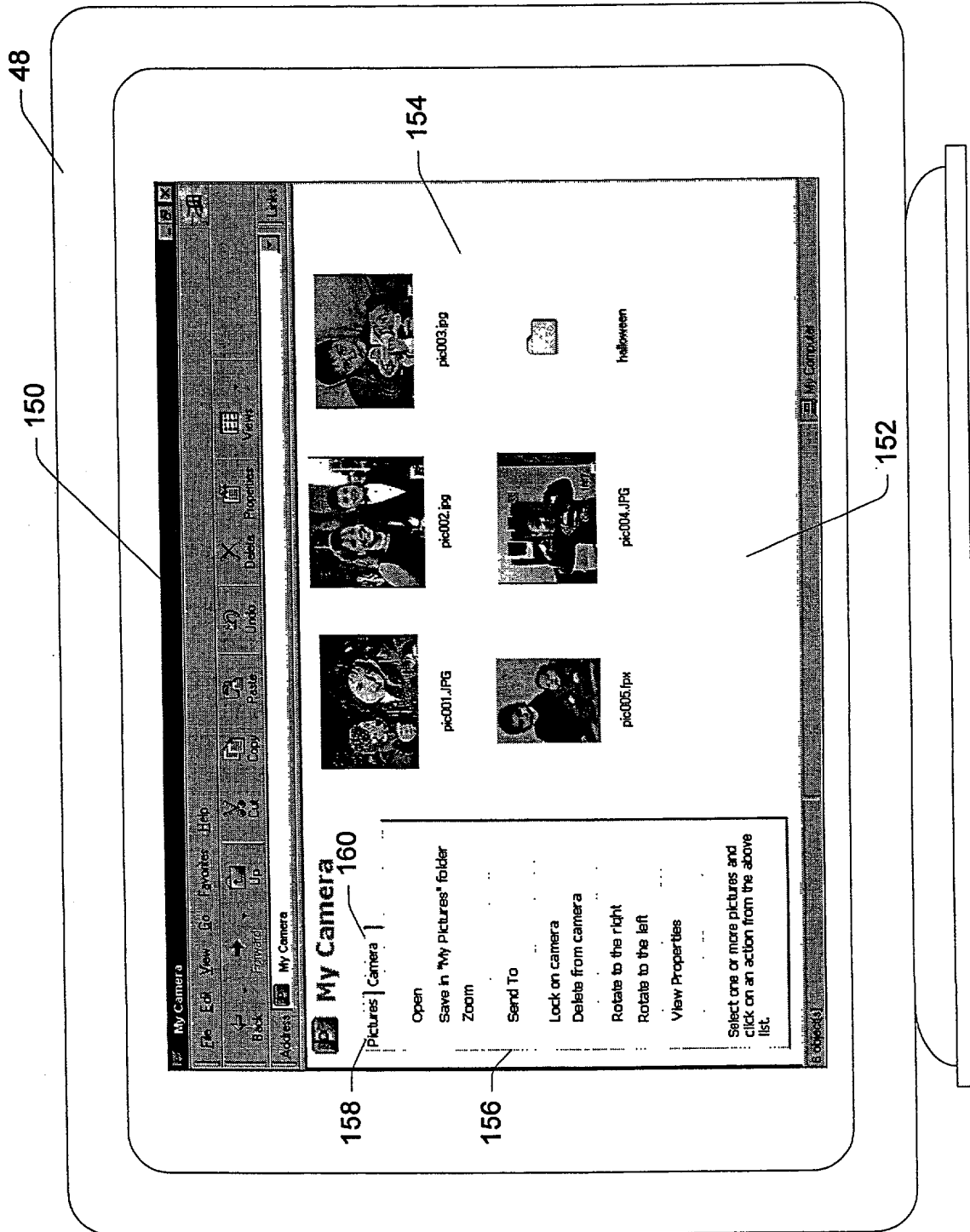


Fig. 6

FIG. 7

48

170

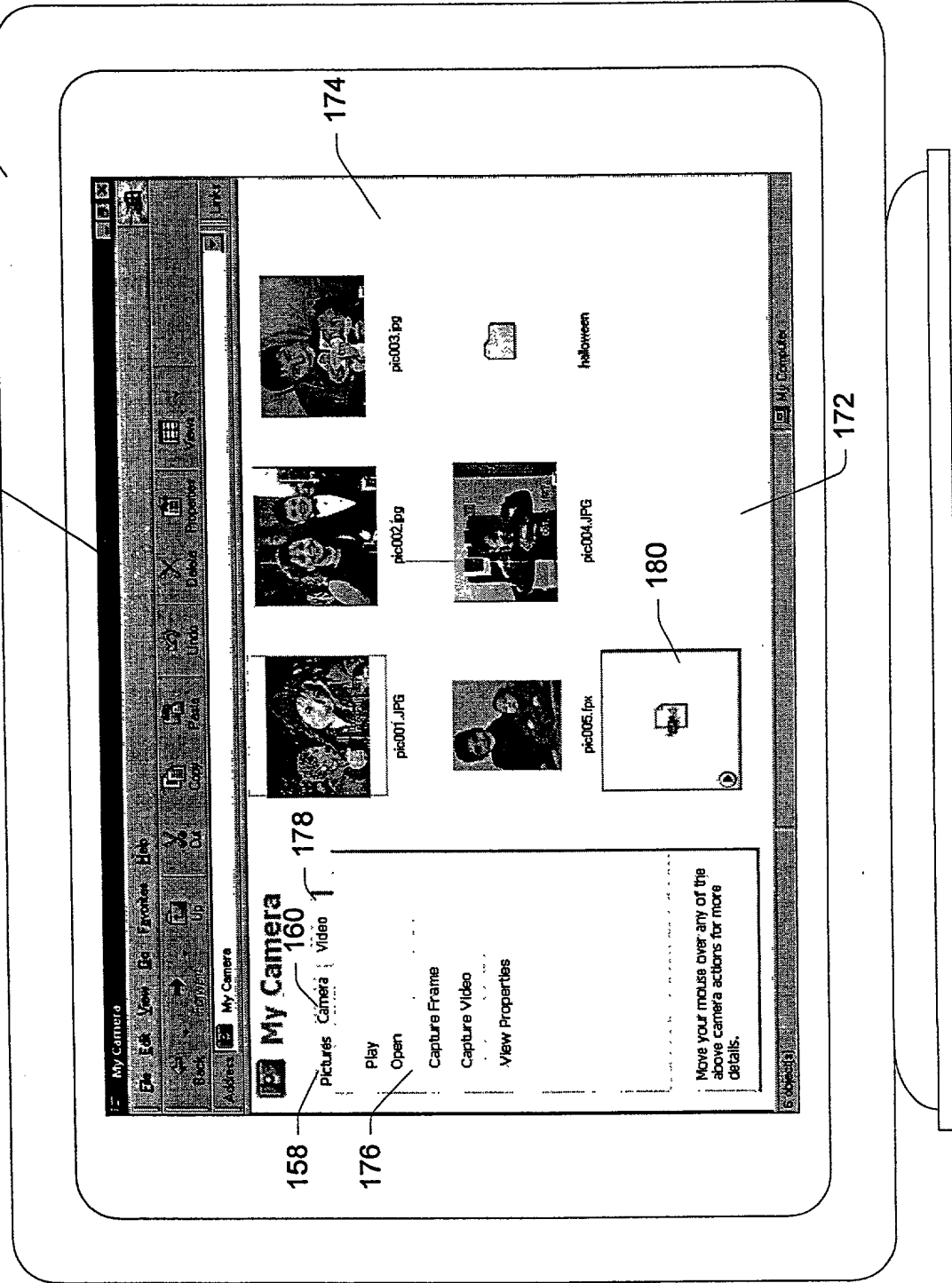


Fig. 7

APPENDIX

ImageIn Device Manager

The ImageIn device manager provides interfaces for enumerating devices, querying properties of installed ImageIn devices and creating ImageIn device objects. The device manager has three objects.

CImageInDevMgr is used to:

- Create an enumerator object, CEnumImageInDevInfo
- Create a device object when given a DeviceID.
- Display UI to let a user choose a device object.
- Display UI to both choose a device and acquire an image from the chosen device.

CEnumImageInDevInfo is used to:

- Enumerate all ImageIn devices on a system. For each device enumerated, a CImageInDevInfo object is returned.

CImageInDevInfo is used to:

- Query device information properties from the ImageIn device. One of the properties, Device ID, can be used by CImageInDevMgr to create a device object.

Device information properties are distinct from normal device properties in that they are read from the registry and can be queried without forcing the device object to be created.

A program using its own methods to select an ImageIn device would do the following:

- Create a CImageInDevMgr object
- Use the IImageInDevMgr interface to create a CEnumImageInDevInfo object
- Use IEnumIMAGEIN_DEV_INFO interface to create a CImageInDevInfo object for each ImageIn device enumerated.
- Use IPropertyStorage interface of CImageInDevInfo object to inspect the device information properties of each device, save the Device ID property from the desired device.
- Use the DeviceID property in IImageInDevMgr interface to create an ImageIn device object.

ImageInDevMgr Interface

EnumImageInDevInfo

```
HRESULT IImageInDevMgr::EnumImageInDevInfo(  
    LONG                lFlags,  
    IEnumIMAGEIN_DEV_INFO** ppIEnum);
```

EnumImageInDevInfo creates a standard enumerator for CImageInDevInfo objects. The enumerator itself is a CEnumImageInDevInfo object that has a single interface IEnumIMAGEIN_DEV_INFO. Applications can use this API to obtain device information properties from available ImageIn devices.

Parameters:

lFlags	Specifies the type of device to enumerate.
ppIEnum	Enumeration interface for CEnumImageInDevInfo objects.

GetImageInDevInfo

```
HRESULT IImageInDevMgr::GetImageInDevInfo(  
    BSTR                bstrDeviceID,  
    IPropertyStorage**  ppIPropStg);
```

Given a device ID GetImageInDevInfo returns an IPropertyStorage interface to CImageInDevInfo objects. This API gives applications a quick way to retrieve device information properties from a stored device ID.

Parameters:

bstrDeviceID,	Device ID string returned from device info object.
ppIPropStg	IPropertyStorage interface to CImageInDevInfo objects.

CreateImageInDev

```
HRESULT IImageInDevMgr::CreateImageInDev(  
    BSTR                bstrDeviceID,  
    IUnknown**          ppDevice);
```

CreateImageInDev creates a device object identified by bstrDeviceID and returns an IUnknown interface to the object. The application can use QueryInterface on this IUnknown to access the other interfaces on a device.

Parameters:

bstrDeviceID	Device ID string returned from device info object.
ppDevice	IUnknown interface pointer returned.

SelectImageInDev

```
HRESULT IImageInDevMgr::SelectImageInDev(  
    LONG                lFlags,  
    IUnknown**          ppDevice);
```

SelectImageInDev displays UI for the selection of ImageIn devices. When the user selects a device, the device object is created and an IUnknown interface is returned. The application can then use QueryInterface on this IUnknown to access the other interfaces on a device. Returns S_FALSE if the user cancels the selection dialog without making a device selection, S_OK if the user makes a device selection or an error value if the method fails.

Parameters:

lFlags	Operation flags
ppDevice	IUnknown interface pointer returned

GetImage

```
HRESULT IImageInDevMgr::GetImage(  
    LONG                lFlags,  
    LONG                lIntent,  
    LONG                cfFormat,  
    STGMEDIUM*         pMedium);
```

GetImage displays device selection UI allowing a user to select a device. If the user selects a device, a device is created and device UI is displayed to capture the image. Image data is returned in the pMedium structure. Image format and default image capture properties may be specified using the lIntent and cfFormat parameters. Returns S_FALSE if the user cancels the selection dialog without making a device selection, S_OK if the user makes a device selection and the data is transferred or an error value if the method fails.

Parameters:

lFlags	Operation flags
lIntent	Intent, default image capture properties
cfFormat	Clipboard format desired by application
pMedium	Image is returned through this medium.

EnumDestinationInfo

```
HRESULT IImageInDevMgr::EnumDestinationInfo(  
    LONG                lFlags,  
    IUnknown*           pDevice,  
    IEnumIMAGEIN_DEST_INFO** ppIEnum);
```

EnumImageInDestInfo creates a standard enumerator for CImageInDestInfo objects. The enumerator itself is a CEnumImageInDestInfo object that has a single interface IEnumIMAGEIN_DEST_INFO. Applications can use this API to obtain device destination properties from an ImageIn device. Destination applications are registered using RegisterDestinationApplication.

Parameters:

lFlags	Enumeration flags.
pDevice	Device object. If null destination applications for all devices are enumerated.
ppIEnum	Enumeration interface for CEnumImageInDestInfo objects.

RegisterDestinationApplication

```
HRESULT IImageInDevMgr::RegisterDestinationApplication(  
    LONG                lFlags,  
    LONG                lIntent,  
    IUnknown*           pDevice,  
    BSTR                bstrEvent,  
    BSTR                bstrAppName,  
    BSTR                bstrCommandLine);
```

This method is called when an application wishes to register as an ImageIn destination.

Parameters:

IFlags	Registration flags.
IIntent	Intent, default image capture properties.
pDevice	Device object. If null, destination application is registered for all devices.
bstrEvent	Optional event name.
bstrAppName	Friendly name of Application. This name will be displayed to the user in UI.
bstrCommandLine	Full path to the executable for this application. Additional command line arguments may be added to this command.

UnregisterDestinationApplication

HRESTUL IImageInDevMgr::UnregisterDestinationApplication

LONG	lFlags,
IUnknown*	pDevice,
BSTR	bstrAppName);

This method is called when an application that has registered an ImageIn destination wishes to be uninstalled or no longer known as an ImageIn destination

Parameters:

IFlags	Operation flags.
pDevice	Device object. If null, destination application is unregistered for all devices.
bstrAppName	Friendly name of Application, used to register application.

IEnumIMAGEIN_DEV_INFO Interface

The IEnumIMAGEIN_DEV_INFO interface is a standard OLE enumeration interface that supports per device enumeration of device information properties.

Next

HRESULT IEnumIMAGEIN_DEV_INFO::Next (
 ULONG **celt,**
 IPropertyStorage** **rgelt,**
 ULONG ***pceltFetched);**

Next returns an array of IPropertyStorage interfaces to CImageInDevInfo objects. Applications can use the returned of IPropertyStorage interfaces to obtain device information properties from available ImageIn devices.

Parameters:

celt	Specifies the number of elements requested.
rgelt	Address of an array in which to return the IPropertyStorage interfaces.
pceltFetched	Address of a value that receives a count of the item identifiers actually returned in <i>rgelt</i> .

Skip

```
HRESULT IEnumIMAGEIN_DEV_INFO::Skip(ULONG celt);
```

Skip skips device objects in the enumeration.

Parameters:

celt	Number of items to skip.
------	--------------------------

Reset

```
HRESULT IEnumIMAGEIN_DEV_INFO::Reset(void);
```

Reset sets the enumeration back to the first device.

Clone

```
HRESULT IEnumIMAGEIN_DEV_INFO::Clone(  
    IEnumIMAGEIN_DEV_INFO **ppIEnum);
```

Clone creates another IEnumIMAGEIN_DEV_INFO enumeration object and returns an interface pointer to it.

Parameters:

ppIEnum	Address that receives a pointer to the new enumeration object.
---------	--

IEnumIMAGEIN_DEST_INFO Interface

The IEnumIMAGEIN_DEST_INFO interface is a standard OLE enumeration interface that supports enumeration of device destination properties.

Next

```
HRESULT IEnumIMAGEIN_DEST_INFO::Next(  
    ULONG          celt,  
    IPropertyStorage** rgelt,  
    ULONG          *pceltFetched);
```

Next returns an array of IPropertyStorage interfaces to CImageInDestInfo objects. Applications can use the returned IPropertyStorage interfaces to obtain destination information properties from available ImageIn devices.

Parameters:

celt	Specifies the number of elements requested.
rgelt	Address of an array in which to return the IPropertyStorage interfaces.
pceltFetched	Address of a value that receives a count of the item identifiers actually returned in <i>rgelt</i> .

Skip

```
HRESULT IEnumIMAGEIN_DEST_INFO::Skip(ULONG celt);
```

Skip skips destination objects in the enumeration.

Parameters:

celt Number of items to skip.

Reset

```
HRESULT IEnumIMAGEIN_DEST_INFO::Reset(void);
```

Reset sets the enumeration back to the first device.

Clone

```
HRESULT IEnumIMAGEIN_DEST_INFO::Clone(  
    IEnumIMAGEIN_DEST_INFO **ppiEnum);
```

Clone creates another IEnumIMAGEIN_DEST_INFO enumeration object and returns an interface pointer to it.

Parameters:

ppiEnum Address that receives a pointer to the new enumeration object.

IPropertyStorage Interface

The IPropertyStorage interface is used to query the device information and destination properties. IPropertyStorage is a standard OLE interface and is documented in the OLE Programmers Reference. All device information properties are read only. Any attempt to modify the device information properties will result in a failure with access denied. The methods of this interface are:

```
HRESULT ReadMultiple(ULONG, const PROPSPEC, PROPVARIANT);  
HRESULT WriteMultiple(ULONG, const PROPSPEC, PROPVARIANT, PROPID);  
HRESULT DeleteMultiple(ULONG, const PROPSPEC);  
HRESULT ReadPropertyNames(ULONG, const PROPID, LPOLESTR);  
HRESULT WritePropertyNames(ULONG, const PROPID, LPOLESTR);  
HRESULT DeletePropertyNames(ULONG, const PROPID);  
HRESULT Commit(DWORD);  
HRESULT Revert(void);  
HRESULT Enum(IEnumSTATPROPSTG**);  
HRESULT SetTimes(FILETIME const *, FILETIME const *, FILETIME const *);  
HRESULT SetClass(REFCLSID);  
HRESULT Stat(STATPROPSETSTG *);
```

Device Information Properties

All ImageIn devices support the following basic device information properties:

Device CLSID	- The CLSID of the ImageIn server for this device.
Unique Device ID	- A device ID which is unique per physical device.

Vendor Description
Device Description
Device Type
Device Port Name
Device Friendly Name
Device Icon Resource
Device Bitmap Resource
Server Name

- The device manufactures name.
- A description of the device.
- A device type constant: scanner, camera...
- Name of the port through which the device is connected.
- A user readable device name.
- Resource file name and ID.
- Resource file name and ID.
- The name of the server where the ImageIn server for this device is running.

ImageIn Scanner Device Object

ImageIn device objects support interfaces for querying and setting device properties, displaying device UI and transferring data. ImageIn devices are required to support a small number of standard interfaces that allow applications to deal with all devices in a common manner and transfer data from the devices in a manner that is native to the (COM) application. Device objects may also support more specialized interfaces to implement custom functions. Since the application has a direct connection to the device object, this architecture does not place any strict limits on the interfaces a device object can export. Practically speaking though, applications must know about an interface for it to be useful.

IScan Interface

ScanDlg

```
HRESULT ScanDlg(  
    LONG                lFlags,  
    LONG                lIntent);
```

ScanDlg presents the system or device UI needed to prepare a device for scanning. If the dialog successfully completes, the scan object will be ready to begin data transfer via the IDataObject or ImageTransfer interfaces. The default image capture properties may be specified using the optional lIntent parameter. Returns S_FALSE if the user cancels the selection dialog without making a device selection, S_OK if the user makes a device selection or an error value if the method fails.

Parameters:

lFlags
lIntent

Operation flags
Intent, default image capture properties

IPropertyStorage Interface

The standard IPropertyStorage interface is used to query and set all scan device properties. IPropertyStorage is a standard OLE interface and is documented in the OLE Programmers Reference. The methods of this interface are:

```
HRESULT ReadMultiple(ULONG, const PROPSPEC, PROPVARIANT);  
HRESULT WriteMultiple(ULONG, const PROPSPEC, PROPVARIANT, PROPID);  
HRESULT DeleteMultiple(ULONG, const PROPSPEC);  
HRESULT ReadPropertyNames(ULONG, const PROPID, LPOLESTR);  
HRESULT WritePropertyNames(ULONG, const PROPID, LPOLESTR);  
HRESULT DeletePropertyNames(ULONG, const PROPID);  
HRESULT Commit(DWORD);  
HRESULT Revert(void);  
HRESULT Enum(IEnumSTATPROPSTG**);  
HRESULT SetTimes(FILETIME const *, FILETIME const *, FILETIME const *);  
HRESULT SetClass(REFCLSID);  
HRESULT Stat(STATPROPSETSTG *);
```

All ImageIn scanner devices support the following basic device properties:

Horizontal Resolution	- The horizontal resolution in DPI.
Vertical Resolution	- The vertical resolution in DPI.
Horizontal Scan Start Position	- The horizontal start position in pixels.
Vertical Scan Start Position	- The vertical start position in pixels.
Horizontal Scan Extent	- The width of the scan in pixels.
Vertical Scan Extent	- The height of the scan in pixels.
Scan Data Type	- A data format specification constant.
Scan Color Depth	- The number of bits per pixel used to specify color.

ImageIn scan devices may support additional device properties depending on hardware configuration.

IDevPropStream Interface

The IDevPropStream interface is used to query/set all current device properties from/to a named, non-volatile, registry based storage. The methods of this interface are:

ReadDevPropStream

HRESULT ReadDevPropStream(BSTR bstrName);

ReadDevPropStream reads a device property stream from the specified value and initializes the device with these properties. The properties are per user and per device.

Parameters:

bstrName	Name of the property stream.
----------	------------------------------

WriteDevPropStream

HRESULT WriteDevPropStream(BSTR bstrName);

WriteDevPropStream writes the current device property stream to the specified value. The properties are per user and per device.

Parameters:

bstrName	Name of the property stream.
----------	------------------------------

ListDevPropStreams

HRESULT ListDevPropStreams(BSTR *pbstrName);

ListDevPropStreams lists the device property stream names present in non-volatile storage. The list is returned in an allocated BSTR which the application must free using SysFreeString.

Parameters:

pbstrName	Pointer to receive a list property streams.
-----------	---

DeleteDevPropStream

HRESULT DeleteDevPropStream(BSTR bstrName);

DeleteDevPropStream deletes a device property stream from non-volatile storage.

Parameters:

bstrName

Name of the property stream.

IDataObject Interface

The IDataObject interface is a standard OLE data transfer mechanism. This interface is used to provide an easy and natural way for applications to transfer data from a scan device. The full IDataObject interface description can be found in the OLE2 Programmer's Reference. The methods of this interface are:

```
HRESULT GetData(LPFORMATETC, LPSTGMEDIUM);
HRESULT GetDataHere(LPFORMATETC, LPSTGMEDIUM);
HRESULT QueryGetData(LPFORMATETC);
HRESULT GetCanonicalFormatEtc(LPFORMATETC, LPFORMATETC);
HRESULT SetData(LPFORMATETC, STGMEDIUM FAR *, BOOL);
HRESULT EnumFormatEtc(DWORD, LPENUMFORMATETC FAR *);
HRESULT DAdvise(FORMATETC FAR *, DWORD, LPADVISESINK, DWORD FAR *);
HRESULT DUnadvise(DWORD);
HRESULT EnumDAdvise(LPENUMSTATDATA FAR *);
```

IImageTransfer Interface

The IImageTransfer is a high performance data transfer interface. This interface uses a shared memory window to transfer data from the device object to the application, eliminating unnecessary data copies during marshalling. For Simplicity, this interface uses the same format negotiation method as IDataObject. IImageTransfer uses two different mechanisms to transfer data.:

Banded Transfers

The device breaks a large image transfer up into smaller transfers, which are performed sequentially into an application-specified buffer.

Data Callback

The device does a single large transfer, while calling back to the application as the transfer progresses.

itAllocateTransferBuffer

```
HRESULT itAllocateTransferBuffer(
    LONG lFlags,
    LONG lSize,
    ULONG hSection,
    ULONG ProcessID);
```

Allocates a buffer to use for scanning data from the device. The buffer is mapped into the address space of both the client and server process.

Parameters:

IFlags	Operation flags.
ISize	Size in bytes of the requested buffer.
hSection	Mapped section handle
ProcessID	Caller process ID

itFreeTransferBuffer

HRESULT itFreeTransferBuffer(void);

Free the buffer created by itAllocateTransferBuffer. This buffer is also freed when the device object is destroyed.

itBeginTransfer

HRESULT itBeginTransfer(
LPFORMATETC pFormatEtc,
LONG lFlags,
IDataCallback* pIDataCallback);

Reset a device object context to begin an IImageTransfer.

Parameters:

pFormatEtc	Format specification.
lFlags	Operation flags.
pIDataCallback	Optional transfer progress notification entry point.

ItGetImage

HRESULT itGetImage(
LONG lFlags,
LONG* plSrcOffset,
LONG* pcbWritten);

Perform an image transfer.

Parameters:

lFlags	Operation flags.
plSrcOffset	Source pointer for transfer.
pcbWritten	Actual number of bytes written during transfer.

itGetImageCB

HRESULT itGetImageCB(LONG lFlags);

Perform an image transfer using a user specified callback.

Parameters:

lFlags	Operation flags.
--------	------------------

itEndTransfer

HRESULT itEndTransfer(LONG lFlags);

Cleanup a device object context after an image transfer.

Parameters:

IFlags

Operation flags.

itQueryGetData

HRESULT itQueryGetData(LPFORMATETC pfe);

Check to see if a device supports a given format in an image transfer.

Parameters:

pfe

Pointer to the FORMATETC structure defining the format, medium, and target device to use for the query.

itEnumFormatEtc

HRESULT itEnumFormatEtc(
 DWORD dw,
 LPENUMFORMATETC* lpEnum);

Create a format enumerator.

Parameters:

dw

Reserved

lpEnum

Indirect pointer to the IEnumFORMATETC interface on the new enumerator object.

ImageIn Camera Device Object

The ImageIn Camera is a hierarchical object. The top-level camera object is used to get basic device information and also to get access to individual images or image folders. The CCameraItem object represents images and Image folders.

ICamera Interface

TakePicture

```
HRESULT TakePicture();
```

The TakePicture method instructs a camera to take a new picture.

GetCameraRootItem

```
HRESULT GetCameraRootItem(ICameraItem** pICameraItem);
```

This method returns an interface to the root CCameraItem object. Using the ICameraItem interface it is possible to enumerate all images on the camera.

Parameters:

ICameraItem	Interface pointer to root camera item
-------------	---------------------------------------

GetCameraItemByHandle

```
HRESULT GetCameraItemByHandle(  
    ULONG ulItemHandle,  
    ICameraItem** pICameraItem);
```

This method returns an ICameraItem interface to the specified CCameraItem object. A handle for a CCameraItem object can only be obtained through the ICameraItem interface.

Parameters:

ulItemHandle	Handle to camera item, previously returned by CameraItem
interface	
ICameraItem	Interface pointer to Camera Item specified by handle

CameraDlg

```
HRESULT CameraDlg(  
    LONG lFlags,  
    LONG lIntent,  
    ICameraItem** pICameraItem);
```

Display the camera UI. UI will select an image on a device and prepare for image transfer. The UI returns a ICameraItem interface to a CCameraItem ready for data transfer. IDataObject or IImageTransfer interfaces are then used to transfer image data.

Parameters:

IFlags	Operational flags
IIntent	High level intent
pICameraItem	Interface pointer to Camera Item selected byDlg

ICameraItem Interface

A CCameraItem object is created for every image and directory on the camera. This is done to support full and flexible hierarchical image storage on cameras. A ICameraItem pointer is returned for each CCameraItem object. ICameraItem methods are defined below.

GetItemType

HRESULT GetItemType(ULONG *pItemType);

GetItemType returns the type of camera item the object represents. The two types of camera items are:

1. ItemTypeFolder - CCameraItem is a folder that may contain other CCameraItems
2. ItemTypeImage - CCameraItem is an image.

Parameters:

pItemType	Item Type returned
-----------	--------------------

GetItemHandle

HRESULT GetItemHandle(ULONG *pItemHandle);

Returns a handle to the CCameraItem. This handle can later be used by the ICamera interface to get an interface pointer back to this object.

Parameters:

PItemHandle Item handle returned. This handle may be used by the GetCameraItemByHandle method of the ICamera interface to obtain an ICameraItem pointer.

OpenImagePropertyStorage

HRESULT OpenImagePropertyStorage(IPropertyStorage** ppIPropertyStorage);

Returns an IPropertyStorage interface to the CCameraItem object.

Parameters:

ppIPropertyStorage	Returned IPropertyStorage interface pointer.
--------------------	--

EnumChildItems

HRESULT EnumChildItems(IEnumCameraItem** ppIEnumCameraItem);

Creates a CEnumCameraItem object and returns an IEnumCameraItem interface to it. This method only works if the camera item is a folder and the folder is not empty, as shown in the figure above.

Parameters:

ppIEnumCameraItem

Pointer to IEnumCameraItem interface.

IEnumCameraItem Interface

IEnumCameraItem is a standard OLE enumerator interface with the usual four methods Next, Skip, Clone and Reset. This enumerator will return an ICameraInterface for each camera item in the current camera folder (that was used in the call to EnumChildItems).

Next

```
HRESULT Next (
    ULONG          celt,
    ICameraItem**  ppICameraItem,
    ULONG*         pceltFetched);
```

Return an ICameraItem interface for each CCameraItem contained by the current folder.

Parameters:

celt	number of camera items to get
ppICameraItem	List of iCameraItem interface pointers returned.
pceltFetched	Number of interface pointers returned

Skip

```
HRESULT Skip(ULONG celt);
```

Skip celt CCameraItem objects.

Parameters:

celt	Number of CcameraItem objects to skip
------	---------------------------------------

Reset

```
HRESULT Reset(void);
```

Begin at the first CCameraItem object.

Parameters: None

Clone

```
HRESULT Clone(IEnumCameraItem **ppIEnumCameraItem );
```

Create a new CEnumCameraItem object and return a IEnumCameraItem interface.

Parameters:

ppIEnumCameraItem	New enumerator created to exactly match this one.
-------------------	---

Camera Download Manager

The camera download manager monitors the connection status of camera devices. When a camera device is connected to the system, the Download Manager will attempt to immediately download all the images from the camera onto the system hard disk. The camera download manager will also maintain a data base of all images previously downloaded from a given camera and not download images that are already cached.

Caching images in this manner has several advantages.

- Camera image download can be very slow, this cache prevents any image from needing to be downloaded more than once.
- Applications do not need a special API to acquire images from a camera, simply wait for the download to complete and load the image from the file system.
- Images are safely and automatically copied to the file system where they can easily be edited, archived, printed or just saved.

In order for the camera download manager to work effectively, camera events must be detectable. For example, it should be easy to detect when a camera is connected or disconnected from the system. This should be detectable without loading the entire camera driver. Also any activity performed directly on the camera (through camera controls) must be reported to software so an accurate internal model of the state of the camera can be kept.

Sample Code

Get an image from an ImageIn device. Use the ImageIn device manager device selection UI and the device image acquisition UI.

```

/*****
 *
 *  GetImageFromImageIn
 *
 *  DESCRIPTION:
 *    Use the ImageIn device manager to select a device and acquire an
 *    image using the device UI.
 *
 *  PARAMETERS:
 *    pszDIBfileName - Name of the DIB file which contain the image data.
 *
 *****/

HRESULT GetImageFromImageIn(LPOLESTR pszDIBfileName)
{
    HRESULT          hr;
    IImageInDevMgr   *pIImageInDevMgr;

    // Get ImageIn device manager object.
    hr = CoCreateInstance(CLSID_ImageInDevMgr,
                          NULL,
                          CLSCTX_LOCAL_SERVER,
                          IID_IImageInDevMgr,
                          (void**) &pIImageInDevMgr);

    if (SUCCEEDED(hr)) {
        STGMEDIUM StgMedium;

        // Fill in the storage medium spec and get the image.
        StgMedium.tymed = TYMED_FILE;
        StgMedium.lpszFileName = pszDIBfileName;
        StgMedium.pUnkForRelease = NULL;
        hr = pIImageInDevMgr->GetImage(0,
                                       0,
                                       CF_DIB,
                                       &StgMedium);

        pIImageInDevMgr->Release();
    }
    return hr;
}

```

Enumerate all installed ImageIn devices and get the device ID of a specific device:

```

/*****
*
* GetDeviceIDfromDescription
*
* DESCRIPTION:
*   Get the unique device ID of an ImageIn device based on it's description.
*
* PARAMETERS:
*   pszDescription   - Requested device description, from device info properties.
*   pszDeviceID      - Buffer for the returned device ID.
*   cbDeviceIDSize   - Size of the returned device ID buffer in bytes.
*
*****/

HRESULT GetDeviceIDfromDescription(
    LPOLESTR pszDescription,
    LPOLESTR pszDeviceID,
    UINT      cchDeviceID)
{
    HRESULT hr;
    IImageInDevMgr *pIImageInDevMgr;

    if (pszDeviceID && cchDeviceID) {
        *pszDeviceID = '\0';
    }
    else {
        return E_INVALIDARG;
    }

    // Get ImageIn device manager object.
    hr = CoCreateInstance(CLSID_ImageInDevMgr,
        NULL,
        CLSCTX_LOCAL_SERVER,
        IID_IImageInDevMgr,
        (void**) &pIImageInDevMgr);

    if (SUCCEEDED(hr)) {
        IEnumIMAGEIN_DEV_INFO *pIEnumIMAGEIN_DEV_INFO;
        PROPSPEC PropSpec[1];
        PROPVARIANT PropVarDesc[1], PropVarID[1];
        UINT cbSize;

        // Get an enumerator for the ImageIn device information.
        hr = pIImageInDevMgr->EnumImageInDevInfo(0, &pIEnumIMAGEIN_DEV_INFO);
        if (SUCCEEDED(hr)) {
            IPropertyStorage *pIPropStg;
            ULONG ul;

            // Enumerate the ImageIn devices, getting a IImageInDevInfo
            // pointer for each. Use this interface to query the registry
            // based properties for each installed device.
            ul = 1;
            while ((hr = pIEnumIMAGEIN_DEV_INFO->Next(1,
                &pIPropStg,
                &ul)) == S_OK) {

                // Read the device description for the current device.
                PropSpec[0].ulKind = PRSPEC_PROPID;
                PropSpec[0].propid = DIP_DEV_DESC;
                hr = pIPropStg->ReadMultiple(1, PropSpec, PropVarDesc);
                if (SUCCEEDED(hr)) {
                    // Test for a description match.
                    if (!wcscmp(PropVarDesc[0].pwszVal, pszDescription)) {
                        // Read the device ID.
                        PropSpec[0].propid = DIP_DEV_ID;
                        hr = pIPropStg->ReadMultiple(1, PropSpec, PropVarID);
                    }
                }
            }
        }
    }
}

```

```

        if (SUCCEEDED(hr)) {
            wcsncpy(pszDeviceID, PropVarID[0].pwszVal, cchDeviceID);
            PropVariantClear(PropVarID);
        }
        PropVariantClear(PropVarDesc);
    }
    pIPropStg->Release();
    if (*pszDeviceID) {
        break;
    }
}
pIEnumIMAGEIN_DEV_INFO->Release();
}
pIImageInDevMgr->Release();
if (!*pszDeviceID) {
    hr = E_FAIL;
}
}
return hr;
}

```